ARP Assignment academic year 2019-2020 (V2.0)

This document describes the specifications of ARP assignment, updated after the pandemic emergency.

The original specifications assumed that all students developed a single piece of code to be exchanged with other students, in order to carry out a cooperative experiment in the lab. As you know, labs will stay closed at least until September. Therefore, it is necessary to change the specifications and transform the cooperative project into an individual one. At the same time I want that what you have already developed will need very small modifications.

The new specifications convert the distributed/cooperative model into a single machine implementation. In the next page you can find the new specifications.

The document reports in yellow all the text changes with respect to the original sheet (V1.0).

Students will produce the following documents:

1. source code with embedded comments (clearness is evaluated)
2. a script for compiling and executing
3. executable form
4. brief description of the source package, how to install, execution instructions
5. brief summary of the experiments and results
6. optional: generated wave plots (see final part of the document).

A repository will be setup in aulaweb for uploading the projects.

These rules are valid also for students who will do the exam in the next dates.

2019 ARP Assignment V2.0

The problem to be solved has the following specifications.

The goal is to simulate a network of multi-process systems, all *identical*, each one running on a machine in the same LAN, connected through sockets, exchanging a data token at a specified and measured *speed*. The number of machines is not given *a priori*. In figure 1 there are 4 machines as an example. They may be more, or less, or they may even reduce to *one single machine* for setup and debugging purposes.

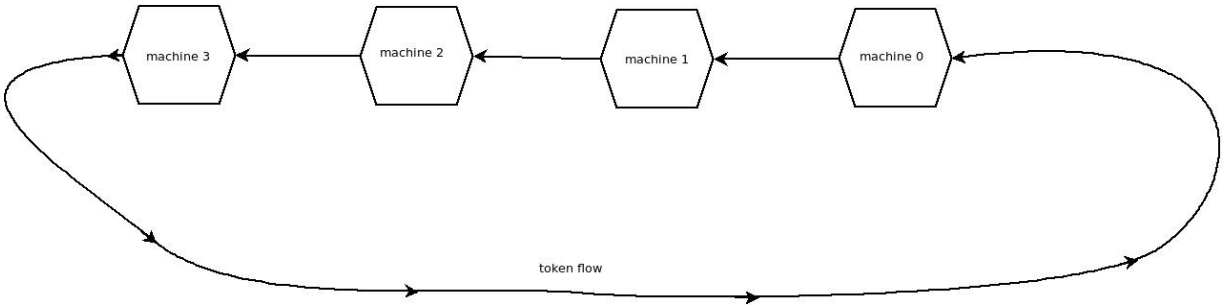

Figure 1

In figure 2 the structure of each machine is defined. Note that a black frame encompasses each machine's multi-process system, whereas a colored frame includes the system components that must developed by each student. Summing up, each student develops all system components of her/his machine, except for one, which is acquired by her/his partner "right side" student; in her/his turn, the student develops a component that is to be given to, and integrated by the"left side" student ("right" and "left" refer to the schema in figure 2).

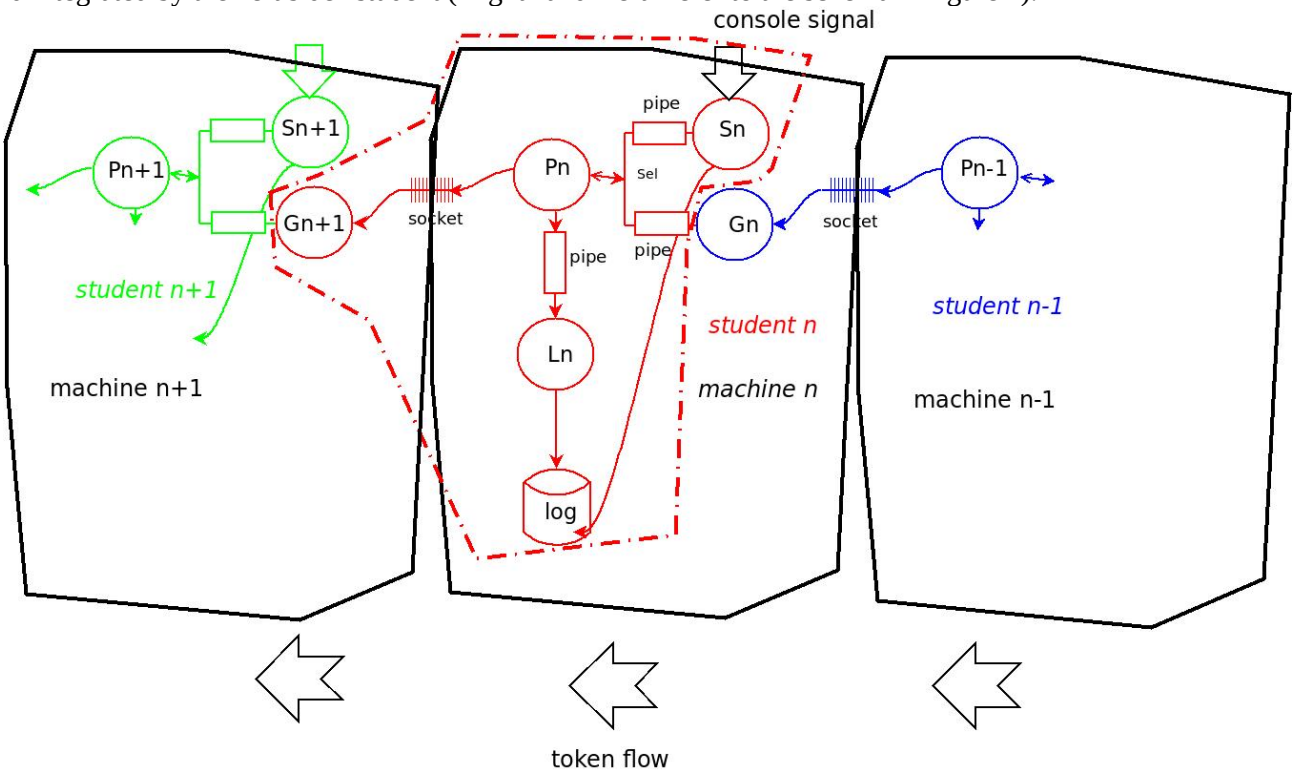

Figure 2

Students will implement on a single machine all processes, simulating a distributed implementation. An artificial delay (it will be called *waiting time*) will be used to mimic the network communication delays.

Glossary.

- Circles are Posix processes.
- Process P receives tokens, computes and sends an updated token after a given delay (see the *waiting time* below).
- Process L logs data end events in a log file.
- Process G receives tokens and dispatches them to P.
- Process S receives console messages as Posix signals.
- Processes S and G are connected to P through non-deterministic pipes (Posix *select*).
- A socket connects P to the ~~partner's~~ G process.

Detailed specifications.

A ***configuration file*** in the machine contains in text format the necessary parameters, and is edited by the user before running the multi-process:

- IP address of the machine and port numbers of processes (where relevant)
- reference frequency (RF) of the generated token wave
- waiting time (in microseconds) applied by process P before sending the updated token (initially: 1,000)
- other relevant data to be read before starting.

Token is:

- a floating point number between -1. and 1, plus
- a time stamp with the absolute system time (by the operating system) of the instant P writes data on socket.

P does the following computation:

new token = received token + DT x (1. - (received token)^2/2) x 2 pi x RF

where DT is the time delay between the reception and delivery time instants (that must be measured by the system). In this way the token series forms a ***sine wave*** of frequency RF. Note that DT takes into account the waiting time written in the configuration file.

Log file
holds a series of text lines in couples:
<timestamp> <from G | from S > <value>
<timestamp> <sent value> (a sample of the wave)

The log file contents is output anytime G receives a suitable request from the console

S receives signals for carrying out the following actions:

- start (receiving tokens, computing, sending tokens)
- stop ((receiving tokens, computing, sending tokens)
- dump log: outputs the contents of the log file, separating the wave from the actions.

Start phase
P, G, S, L start using the parameters stored in the configuration file.

~~***The process G*** is given by the "right side" student in executable form, and is run using the *exec* syscall.~~

Main syscalls involved:

fork(), exec()
pipe()
select()
read() write()
socket()
signal()
and related ones.

Organization and goal

Students should setup their own multi-process systems by implementing all components in a single machine.
The system components to implement are those in the red frame of figure 2 Referring to this figure, **processes Gn and Gn+1 coincide** (only one process), **as well as Pn and Pn-1.**
The operating system is Ubuntu 16.04 LTS or later versions.
~~The final test will be carried out by triple or quad groups of randomly selected students.~~
The code must be - mandatorily - such as all processes might run on separate machines.
Experiments will start with a ***waiting time*** value equal to 1,000 microseconds and a ***sine wave frequency*** **RF** equal to 1; then, students will change those values in order to find the **maximum RF frequency and the minimum waiting time compatible with their implementation.**
A graphical interface for inspecting the generated wave is strongly recommended (e.g. using matlab or excel or another app on the log file). Please give the processor and opsys types.